EDA Data Transformation

November 27, 2024

1 Data Transformation

1. Data deduplication involves the identification of duplicates and their removal.
2. Key restructuring involves transforming any keys with built-in meanings to the generic keys.
3. Data cleansing involves extracting words and deleting out-of-date, inaccurate, and incomplete information from the source language without extracting the meaning or information to enhance the accuracy of the source data.
4. Data validation is a process of formulating rules or algorithms that help in validating different types of data against some known issues.
5. Format revisioning involves converting from one format to another.
6. Data derivation consists of creating a set of rules to generate more information from the data source.
7. Data aggregation involves searching, extracting, summarizing, and preserving important information in different types of reporting systems.
8. Data integration involves converting different data types and merging them into a common structure or schema.
9. Data filtering involves identifying information relevant to any particular user.
10. Data joining involves establishing a relationship between two or more tables.

[1]:
```
import pandas as pd
import numpy as np
```

2 Combining dataframes

[2]:
```
dataFrame1 =  pd.DataFrame({ 'StudentID': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19,␣
↪21, 23, 25, 27, 29], 'Score' : [89, 39, 50, 97, 22, 66, 31, 51, 71, 91, 56,␣
↪32, 52, 73, 92]})
dataFrame2 =  pd.DataFrame({'StudentID': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20,␣
↪22, 24, 26, 28, 30], 'Score': [98, 93, 44, 77, 69, 56, 31, 53, 78, 93, 56,␣
↪77, 33, 56, 27]})

# In the dataset above, the first column contains information about student␣
↪identifier and the second column contains their respective scores in any␣
↪subject. The structure of the dataframes is same in the bothe case. In this␣
↪case, we would need to concatenate both of them.
```

[3]:
```python
# We can do that by using Pandas concat() method.

dataframe = pd.concat([dataFrame1, dataFrame2], ignore_index=True)
dataframe
```

[3]:

	StudentID	Score
0	1	89
1	3	39
2	5	50
3	7	97
4	9	22
5	11	66
6	13	31
7	15	51
8	17	71
9	19	91
10	21	56
11	23	32
12	25	52
13	27	73
14	29	92
15	2	98
16	4	93
17	6	44
18	8	77
19	10	69
20	12	56
21	14	31
22	16	53
23	18	78
24	20	93
25	22	56
26	24	77
27	26	33
28	28	56
29	30	27

The argument ignore_index creates new index and its absense keeps the original indices. Note, we combined the dataframes along axis=0, that is to say, we combined together along same direction. What if we want to combine both side by side. Then we have to specify axis = 1. Check the output and see the difference.

[4]:
```python
pd.concat([dataFrame1, dataFrame2], axis=1)
```

[4]:

	StudentID	Score	StudentID	Score
0	1	89	2	98
1	3	39	4	93
2	5	50	6	44

```
3           7      97          8      77
4           9      22         10      69
5          11      66         12      56
6          13      31         14      31
7          15      51         16      53
8          17      71         18      78
9          19      91         20      93
10         21      56         22      56
11         23      32         24      77
12         25      52         26      33
13         27      73         28      56
14         29      92         30      27
```

3 Merging

In the first example, you received two files for same subject. Now, consider the use case where you are teaching two courses. So, you will get two dataframes from each sections: two for Software engieering course and another two for Introduction to Machine learning course. Check the figure given below:

[5]:
```
df1SE =  pd.DataFrame({ 'StudentID': [9, 11, 13, 15, 17, 19, 21, 23, 25, 27,␣
 ↪29], 'ScoreSE' : [22, 66, 31, 51, 71, 91, 56, 32, 52, 73, 92]})
df2SE =  pd.DataFrame({'StudentID': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22,␣
 ↪24, 26, 28, 30], 'ScoreSE': [98, 93, 44, 77, 69, 56, 31, 53, 78, 93, 56, 77,␣
 ↪33, 56, 27]})

df1ML =  pd.DataFrame({ 'StudentID': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21,␣
 ↪23, 25, 27, 29], 'ScoreML' : [39, 49, 55, 77, 52, 86, 41, 77, 73, 51, 86,␣
 ↪82, 92, 23, 49]})
df2ML =  pd.DataFrame({'StudentID': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],␣
 ↪'ScoreML': [93, 44, 78, 97, 87, 89, 39, 43, 88, 78]})
```

As you can see in the dataset above, you have two dataframes for each subjects. So the first task would be to concatenate these two subjects into one. Secondly, these students have taken Introduction to Machine Learning course as well. So, we need to merge these score into the same dataframes. There are several ways to do this. Let us explore some options.

[6]:
```
# Option 1
dfSE = pd.concat([df1SE, df2SE], ignore_index=True)
dfML = pd.concat([df1ML, df2ML], ignore_index=True)

df = pd.concat([dfML, dfSE], axis=1)
df
```

[6]:
```
    StudentID  ScoreML  StudentID  ScoreSE
0         1.0     39.0          9       22
1         3.0     49.0         11       66
```

2	5.0	55.0	13	31
3	7.0	77.0	15	51
4	9.0	52.0	17	71
5	11.0	86.0	19	91
6	13.0	41.0	21	56
7	15.0	77.0	23	32
8	17.0	73.0	25	52
9	19.0	51.0	27	73
10	21.0	86.0	29	92
11	23.0	82.0	2	98
12	25.0	92.0	4	93
13	27.0	23.0	6	44
14	29.0	49.0	8	77
15	2.0	93.0	10	69
16	4.0	44.0	12	56
17	6.0	78.0	14	31
18	8.0	97.0	16	53
19	10.0	87.0	18	78
20	12.0	89.0	20	93
21	14.0	39.0	22	56
22	16.0	43.0	24	77
23	18.0	88.0	26	33
24	20.0	78.0	28	56
25	NaN	NaN	30	27

[7]:
```
# Option 2
dfSE = pd.concat([df1SE, df2SE], ignore_index=True)
dfML = pd.concat([df1ML, df2ML], ignore_index=True)

df = dfSE.merge(dfML, how='inner')
df

# Here, you will perform inner join with each dataframe. That is to say, if an␣
 ↪item exists on the both dataframe, will be included in the new dataframe.␣
 ↪This means, we will get the list of students who are appearing in both the␣
 ↪courses.
```

[7]:

	StudentID	ScoreSE	ScoreML
0	9	22	52
1	11	66	86
2	13	31	41
3	15	51	77
4	17	71	73
5	19	91	51
6	21	56	86
7	23	32	82
8	25	52	92

9	27	73	23
10	29	92	49
11	2	98	93
12	4	93	44
13	6	44	78
14	8	77	97
15	10	69	87
16	12	56	89
17	14	31	39
18	16	53	43
19	18	78	88
20	20	93	78

[8]:
```
# Option 3
dfSE = pd.concat([df1SE, df2SE], ignore_index=True)
dfML = pd.concat([df1ML, df2ML], ignore_index=True)

df = dfSE.merge(dfML, how='left')
df
```

[8]:

	StudentID	ScoreSE	ScoreML
0	9	22	52.0
1	11	66	86.0
2	13	31	41.0
3	15	51	77.0
4	17	71	73.0
5	19	91	51.0
6	21	56	86.0
7	23	32	82.0
8	25	52	92.0
9	27	73	23.0
10	29	92	49.0
11	2	98	93.0
12	4	93	44.0
13	6	44	78.0
14	8	77	97.0
15	10	69	87.0
16	12	56	89.0
17	14	31	39.0
18	16	53	43.0
19	18	78	88.0
20	20	93	78.0
21	22	56	NaN
22	24	77	NaN
23	26	33	NaN
24	28	56	NaN
25	30	27	NaN

[9]:
```
# Option 4
dfSE = pd.concat([df1SE, df2SE], ignore_index=True)
dfML = pd.concat([df1ML, df2ML], ignore_index=True)

df = dfSE.merge(dfML, how='right')
df
```

[9]:

	StudentID	ScoreSE	ScoreML
0	1	NaN	39
1	3	NaN	49
2	5	NaN	55
3	7	NaN	77
4	9	22.0	52
5	11	66.0	86
6	13	31.0	41
7	15	51.0	77
8	17	71.0	73
9	19	91.0	51
10	21	56.0	86
11	23	32.0	82
12	25	52.0	92
13	27	73.0	23
14	29	92.0	49
15	2	98.0	93
16	4	93.0	44
17	6	44.0	78
18	8	77.0	97
19	10	69.0	87
20	12	56.0	89
21	14	31.0	39
22	16	53.0	43
23	18	78.0	88
24	20	93.0	78

[10]:
```
# Option 5
dfSE = pd.concat([df1SE, df2SE], ignore_index=True)
dfML = pd.concat([df1ML, df2ML], ignore_index=True)

df = dfSE.merge(dfML, how='outer')
df
```

[10]:

	StudentID	ScoreSE	ScoreML
0	1	NaN	39.0
1	2	98.0	93.0
2	3	NaN	49.0
3	4	93.0	44.0
4	5	NaN	55.0

```
5             6     44.0     78.0
6             7      NaN     77.0
7             8     77.0     97.0
8             9     22.0     52.0
9            10     69.0     87.0
10           11     66.0     86.0
11           12     56.0     89.0
12           13     31.0     41.0
13           14     31.0     39.0
14           15     51.0     77.0
15           16     53.0     43.0
16           17     71.0     73.0
17           18     78.0     88.0
18           19     91.0     51.0
19           20     93.0     78.0
20           21     56.0     86.0
21           22     56.0      NaN
22           23     32.0     82.0
23           24     77.0      NaN
24           25     52.0     92.0
25           26     33.0      NaN
26           27     73.0     23.0
27           28     56.0      NaN
28           29     92.0     49.0
29           30     27.0      NaN
```

[88]:

```
#Merging on index
left1 = pd.DataFrame({'key': ['apple','ball','apple', 'apple','ball', 'cat'],␣
 ↪'value': range(6)})
right1 = pd.DataFrame({'group_val': [33.4, 5]}, index=['apple', 'ball'])
left1
```

[88]:

```
     key  value
0  apple      0
1   ball      1
2  apple      2
3  apple      3
4   ball      4
5    cat      5
```

[87]:

```
right1
```

[87]:

```
       group_val
apple       33.4
ball         5.0
```

[89]:
```python
#let's try merging using an inner join, which is the default type of merge.
df = pd.merge(left1, right1, left_on='key', right_index=True)
df
```

[89]:

	key	value	group_val
0	apple	0	33.4
1	ball	1	5.0
2	apple	2	33.4
3	apple	3	33.4
4	ball	4	5.0

[90]:
```python
# let's try merging using an outer join, as follows
df = pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
df
```

[90]:

	key	value	group_val
0	apple	0	33.4
2	apple	2	33.4
3	apple	3	33.4
1	ball	1	5.0
4	ball	4	5.0
5	cat	5	NaN

3.1 Reshaping and pivoting

[91]:
```python
data = np.arange(15).reshape((3,5))
indexers = ['Rainfall', 'Humidity', 'Wind']
dframe1 = pd.DataFrame(data, index=indexers, columns=['Bergen','Oslo',␣
↪'Trondheim', 'Stavanger', 'Kristiansand'])
dframe1
```

[91]:

	Bergen	Oslo	Trondheim	Stavanger	Kristiansand
Rainfall	0	1	2	3	4
Humidity	5	6	7	8	9
Wind	10	11	12	13	14

[92]:
```python
# using the stack() method on the preceding dframe1, we can pivot the columns␣
↪into rows to produce a series
stacked = dframe1.stack()
stacked
```

[92]:
```
Rainfall  Bergen          0
          Oslo            1
          Trondheim       2
          Stavanger       3
          Kristiansand    4
Humidity  Bergen          5
```

```
            Oslo            6
            Trondheim       7
            Stavanger       8
            Kristiansand    9
Wind        Bergen         10
            Oslo           11
            Trondheim      12
            Stavanger      13
            Kristiansand   14
dtype: int64
```

[93]:
```
# The preceding series stored unstacked in the variable can be rearranged into␣
↪a dataframe using the unstack() method:
stacked.unstack()
```

[93]:
```
           Bergen  Oslo  Trondheim  Stavanger  Kristiansand
Rainfall        0     1          2          3             4
Humidity        5     6          7          8             9
Wind           10    11         12         13            14
```

[94]:
```
# Note that there is a chance that unstacking will create missing data if all␣
↪the values are not present in each of the sub-groups. ex:
series1 = pd.Series([000, 111, 222, 333], index=['zeros','ones', 'twos',␣
↪'threes'])
series2 = pd.Series([444, 555, 666], index=['fours', 'fives', 'sixes'])
frame2 = pd.concat([series1, series2], keys=['Number1', 'Number2'])

frame2.unstack()
```

[94]:
```
         fives  fours   ones  sixes  threes   twos  zeros
Number1    NaN    NaN  111.0    NaN   333.0  222.0    0.0
Number2  555.0  444.0    NaN  666.0     NaN    NaN    NaN
```

3.2 Transformatio techniques

3.2.1 Performing data deduplication

[95]:
```
frame3 = pd.DataFrame({'column 1': ['Looping'] * 3 + ['Functions'] * 4, 'column␣
↪2': [10, 10, 22, 23, 23, 24, 24]})
frame3
```

[95]:
```
    column 1  column 2
0    Looping        10
1    Looping        10
2    Looping        22
3  Functions        23
4  Functions        23
```

```
5   Functions        24
6   Functions        24
```

[96]:
```
frame3.duplicated()
```

[96]:
```
0    False
1     True
2    False
3    False
4     True
5    False
6     True
dtype: bool
```

[97]:
```
frame4 = frame3.drop_duplicates()
frame4
```

[97]:
```
    column 1  column 2
0    Looping        10
2    Looping        22
3  Functions        23
5  Functions        24
```

[98]:
```
frame3['column 3'] = range(7)
frame5 = frame3.drop_duplicates(['column 2'])
frame5
```

[98]:
```
    column 1  column 2  column 3
0    Looping        10         0
2    Looping        22         2
3  Functions        23         3
5  Functions        24         5
```

3.3 Replacing values

[100]:
```
import numpy as np
replaceFrame = pd.DataFrame({'column 1': [200., 3000., -786., 3000., 234., 444.
 ↪, -786., 332., 3332. ], 'column 2': range(9)})
replaceFrame.replace(to_replace =-786, value= np.nan)
```

[100]:
```
   column 1  column 2
0     200.0         0
1    3000.0         1
2       NaN         2
3    3000.0         3
4     234.0         4
5     444.0         5
```

```
6      NaN        6
7    332.0        7
8   3332.0        8
```

[101]:
```
replaceFrame = pd.DataFrame({'column 1': [200., 3000., -786., 3000., 234., 444.
 ↪, -786., 332., 3332. ], 'column 2': range(9)})
replaceFrame.replace(to_replace =[-786, 0], value= [np.nan, 2])
```

[101]:
```
   column 1  column 2
0     200.0         2
1    3000.0         1
2       NaN         2
3    3000.0         3
4     234.0         4
5     444.0         5
6       NaN         6
7     332.0         7
8    3332.0         8
```

3.4 Handling missing data

Whenever there are missing values, a NaN value is used, which indicates that there is no value specified for that particular index. There could be several reasons why a value could be NaN: 1. It can happen when data is retrieved from an external source and there are some incomplete values in the dataset. 2. It can also happen when we join two different datasets and some values are not matched. 3. Missing values due to data collection errors. 4. When the shape of data changes, there are new additional rows or columns that are not determined. 5. Reindexing of data can result in incomplete data.

[102]:
```
data = np.arange(15, 30).reshape(5, 3)
dfx = pd.DataFrame(data, index=['apple', 'banana', 'kiwi', 'grapes', 'mango'],␣
 ↪columns=['store1', 'store2', 'store3'])
dfx
```

[102]:
```
        store1  store2  store3
apple       15      16      17
banana      18      19      20
kiwi        21      22      23
grapes      24      25      26
mango       27      28      29
```

[103]:
```
dfx['store4'] = np.nan
dfx.loc['watermelon'] = np.arange(15, 19)
dfx.loc['oranges'] = np.nan
dfx['store5'] = np.nan
dfx['store4']['apple'] = 20.
dfx
```

```
/tmp/ipykernel_44487/320494991.py:5: FutureWarning: ChainedAssignmentError:
behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in
certain cases, but when using Copy-on-Write (which will become the default
behaviour in pandas 3.0) this will never work to update the original DataFrame
or Series, because the intermediate object on which we are setting values will
behave as a copy.
A typical example is when you are setting values in a column of a DataFrame,
like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in
a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  dfx['store4']['apple'] = 20.
```

[103]:

	store1	store2	store3	store4	store5
apple	15.0	16.0	17.0	20.0	NaN
banana	18.0	19.0	20.0	NaN	NaN
kiwi	21.0	22.0	23.0	NaN	NaN
grapes	24.0	25.0	26.0	NaN	NaN
mango	27.0	28.0	29.0	NaN	NaN
watermelon	15.0	16.0	17.0	18.0	NaN
oranges	NaN	NaN	NaN	NaN	NaN

[ ]:

```

```

[105]:

```python
dfx.isnull()
```

[105]:

	store1	store2	store3	store4	store5
apple	False	False	False	False	True
banana	False	False	False	True	True
kiwi	False	False	False	True	True
grapes	False	False	False	True	True
mango	False	False	False	True	True
watermelon	False	False	False	False	True
oranges	True	True	True	True	True

[106]:

```python
# We can use the sum() method to count the number of NaN values in each store.
dfx.isnull().sum()
```

[106]:

```
store1    1
store2    1
```

```
store3    1
store4    5
store5    7
dtype: int64
```

[107]:
```python
dfx.isnull().sum().sum()
```

[107]:
```
15
```

[109]:
```python
# instead of counting the number of missing values, we can count the numberof␣
↪reported values:
dfx.count()
```

[109]:
```
store1    6
store2    6
store3    6
store4    2
store5    0
dtype: int64
```

3.5 Dropping missing values

[111]:
```python
#determine the null values
dfx.store4[dfx.store4.notnull()]
```

[111]:
```
apple         20.0
watermelon    18.0
Name: store4, dtype: float64
```

[112]:
```python
# Now, we can use the dropna() method to remove the rows:
dfx.store4.dropna()
```

[112]:
```
apple         20.0
watermelon    18.0
Name: store4, dtype: float64
```

[116]:
```python
dfx
```

[116]:

	store1	store2	store3	store4	store5
apple	15.0	16.0	17.0	20.0	NaN
banana	18.0	19.0	20.0	NaN	NaN
kiwi	21.0	22.0	23.0	NaN	NaN
grapes	24.0	25.0	26.0	NaN	NaN
mango	27.0	28.0	29.0	NaN	NaN
watermelon	15.0	16.0	17.0	18.0	NaN
oranges	NaN	NaN	NaN	NaN	NaN

[117]:
```python
# Note that the dropna() method just returns a copy of the dataframe by␣
↪dropping the rows with NaN. The original dataframe is not changed.
# If dropna() is applied to the entire dataframe, then it will drop all the␣
↪rows from the dataframe, because there is at least one NaN value in our␣
↪dataframe:
# dfx.dropna()

# Dropping by rows
dfx.dropna(how='all')
```

[117]:

	store1	store2	store3	store4	store5
apple	15.0	16.0	17.0	20.0	NaN
banana	18.0	19.0	20.0	NaN	NaN
kiwi	21.0	22.0	23.0	NaN	NaN
grapes	24.0	25.0	26.0	NaN	NaN
mango	27.0	28.0	29.0	NaN	NaN
watermelon	15.0	16.0	17.0	18.0	NaN

[118]:
```python
# Dropping by columns
dfx.dropna(how='all', axis=1)
```

[118]:

	store1	store2	store3	store4
apple	15.0	16.0	17.0	20.0
banana	18.0	19.0	20.0	NaN
kiwi	21.0	22.0	23.0	NaN
grapes	24.0	25.0	26.0	NaN
mango	27.0	28.0	29.0	NaN
watermelon	15.0	16.0	17.0	18.0
oranges	NaN	NaN	NaN	NaN

3.5.1 Mathematical operations with NaN

Note the following things: 1. When a NumPy function encounters NaN values, it returns NaN. 2. Pandas, on the other hand, ignores the NaN values and moves ahead with processing. When performing the sum operation, NaN is treated as 0. If all the values are NaN, the result is also NaN.

[120]:
```python
ar1 = np.array([100, 200, np.nan, 300])
ser1 = pd.Series(ar1)
ser1
```

[120]:
```
0    100.0
1    200.0
2      NaN
3    300.0
dtype: float64
```

[122]:
```
ar1.mean(), ser1.mean()
```

[122]:
```
(nan, 200.0)
```

3.6 Filling missing values

[123]:
```
filledDf = dfx.fillna(0)
filledDf
```

[123]:

	store1	store2	store3	store4	store5
apple	15.0	16.0	17.0	20.0	0.0
banana	18.0	19.0	20.0	0.0	0.0
kiwi	21.0	22.0	23.0	0.0	0.0
grapes	24.0	25.0	26.0	0.0	0.0
mango	27.0	28.0	29.0	0.0	0.0
watermelon	15.0	16.0	17.0	18.0	0.0
oranges	0.0	0.0	0.0	0.0	0.0

[124]:
```
dfx.mean()
```

[124]:
```
store1    20.0
store2    21.0
store3    22.0
store4    19.0
store5     NaN
dtype: float64
```

[125]:
```
filledDf.mean()
```

[125]:
```
store1    17.142857
store2    18.000000
store3    18.857143
store4     5.428571
store5     0.000000
dtype: float64
```

[128]:
```
# Backward and forward filling
# Here, from the forward-filling technique, the last known value is 20 and␣
 ↪hence the rest of the NaN values are replaced by it.
dfx.store4.fillna(method='ffill')
```

```
/tmp/ipykernel_44487/1459500098.py:3: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
instead.
  dfx.store4.fillna(method='ffill')
```

[128]:
```
apple         20.0
banana        20.0
kiwi          20.0
grapes        20.0
mango         20.0
watermelon    18.0
oranges       18.0
Name: store4, dtype: float64
```

[129]:
```
#The direction of the fill can be changed by changing method='bfill'. Check the␣
↪following example:ser3 = pd.Series([100, np.nan, np.nan, np.nan, 292])
ser3.interpolate()
dfx.store4.fillna(method='bfill')
```

```
/tmp/ipykernel_44487/1648656597.py:2: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
instead.
  dfx.store4.fillna(method='bfill')
```

[129]:
```
apple         20.0
banana        18.0
kiwi          18.0
grapes        18.0
mango         18.0
watermelon    18.0
oranges        NaN
Name: store4, dtype: float64
```

[131]:
```
# Interpolating missing values
ser3 = pd.Series([100, np.nan, np.nan, np.nan, 292])
ser3.interpolate()
```

[131]:
```
0    100.0
1    148.0
2    196.0
3    244.0
4    292.0
dtype: float64
```

3.6.1 Renaming axis indexes

[132]:
```
dframe1.index = dframe1.index.map(str.upper)
dframe1
```

[132]:

	Bergen	Oslo	Trondheim	Stavanger	Kristiansand
RAINFALL	0	1	2	3	4
HUMIDITY	5	6	7	8	9

```
WIND           10    11         12         13             14
```

```
[133]: dframe1.rename(index=str.title, columns=str.upper)
```

```
[133]:           BERGEN  OSLO  TRONDHEIM  STAVANGER  KRISTIANSAND
Rainfall       0     1          2          3             4
Humidity       5     6          7          8             9
Wind          10    11         12         13            14
```

3.7 Outlier detection and filtering

Outliers are data points that diverge from other observations for several reasons. During the EDA phase, one of our common tasks is to detect and filter these outliers. The main reason for this detection and filtering of outliers is that the presence of such outliers can cause serious issues in statistical analysis.

```
[11]: df = pd.read_csv('https://raw.githubusercontent.com/PacktPublishing/
       ↪hands-on-exploratory-data-analysis-with-python/master/Chapter%204/sales.csv')
      df.head(10)
```

```
[11]:     Account           Company  Order               SKU        Country  Year  \
0  123456779         Kulas Inc  99985   s9-supercomputer          Aruba  1981
1  123456784            GitHub  99986   s4-supercomputer         Brazil  2001
2  123456782         Kulas Inc  99990  s10-supercomputer     Montserrat  1973
3  123456783         My SQ Man  99999   s1-supercomputer    El Salvador  2015
4  123456787        ABC Dogma  99996   s6-supercomputer         Poland  1970
5  123456778  Super Sexy Dingo  99996   s9-supercomputer     Costa Rica  2004
6  123456783        ABC Dogma  99981  s11-supercomputer          Spain  2006
7  123456785        ABC Dogma  99998   s9-supercomputer        Belarus  2015
8  123456778        Loolo INC  99997   s8-supercomputer      Mauritius  1999
9  123456775         Kulas Inc  99997   s7-supercomputer  French Guiana  2004

   Quantity  UnitPrice  transactionComplete
0      5148        545                False
1      3262        383                False
2      9119        407                 True
3      3097        615                False
4      3356         91                 True
5      2474        136                 True
6      4081        195                False
7      6576        603                False
8      2460         36                False
9      1831        664                 True
```

```
[12]: #@title Default title text
      #Add new colum that is the total price based on the quantity and the unit price
```

```python
df['TotalPrice'] = df['UnitPrice'] * df['Quantity']
df.head(10)
```

[12]:

```
     Account            Company  Order              SKU         Country  Year  \
0  123456779          Kulas Inc  99985   s9-supercomputer           Aruba  1981
1  123456784             GitHub  99986   s4-supercomputer          Brazil  2001
2  123456782          Kulas Inc  99990  s10-supercomputer      Montserrat  1973
3  123456783          My SQ Man  99999   s1-supercomputer     El Salvador  2015
4  123456787         ABC Dogma  99996   s6-supercomputer          Poland  1970
5  123456778  Super Sexy Dingo  99996   s9-supercomputer      Costa Rica  2004
6  123456783         ABC Dogma  99981  s11-supercomputer           Spain  2006
7  123456785         ABC Dogma  99998   s9-supercomputer         Belarus  2015
8  123456778         Loolo INC  99997   s8-supercomputer       Mauritius  1999
9  123456775          Kulas Inc  99997   s7-supercomputer   French Guiana  2004

   Quantity  UnitPrice  transactionComplete  TotalPrice
0      5148        545                False     2805660
1      3262        383                False     1249346
2      9119        407                 True     3711433
3      3097        615                False     1904655
4      3356         91                 True      305396
5      2474        136                 True      336464
6      4081        195                False      795795
7      6576        603                False     3965328
8      2460         36                False       88560
9      1831        664                 True     1215784
```

[13]:

```python
df['Company'].value_counts()
```

[13]:

```
Company
My SQ Man                   869
Kirlosker Service Center    863
Will LLC                    862
ABC Dogma                   848
Kulas Inc                   840
Gen Power                   836
Name IT                     836
Super Sexy Dingo            828
GitHub                      823
Loolo INC                   822
SAS Web Tec                 798
Pryianka Ji                 775
Name: count, dtype: int64
```

[14]:

```python
df.describe()
```

[14]:
```
             Account         Order          Year      Quantity     UnitPrice  \
count  1.000000e+04  10000.000000  10000.000000  10000.000000  10000.000000
mean   1.234568e+08  99989.562900   1994.619800   4985.447300    355.866600
std    5.741156e+00      5.905551     14.432771   2868.949686    201.378478
min    1.234568e+08  99980.000000   1970.000000      0.000000     10.000000
25%    1.234568e+08  99985.000000   1982.000000   2505.750000    181.000000
50%    1.234568e+08  99990.000000   1995.000000   4994.000000    356.000000
75%    1.234568e+08  99995.000000   2007.000000   7451.500000    531.000000
max    1.234568e+08  99999.000000   2019.000000   9999.000000    700.000000

         TotalPrice
count  1.000000e+04
mean   1.773301e+06
std    1.540646e+06
min    0.000000e+00
25%    5.003370e+05
50%    1.335698e+06
75%    2.711653e+06
max    6.841580e+06
```

3.8 Reshaping with Hierarchical Indexing

[15]:
```python
data = np.arange(15).reshape((3,5))
indexers = ['Rainfall', 'Humidity', 'Wind']
dframe1 = pd.DataFrame(data, index=indexers, columns=['Bergen', 'Oslo',
 ↪'Trondheim', 'Stavanger', 'Kristiansand'])
dframe1
```

[15]:
```
          Bergen  Oslo  Trondheim  Stavanger  Kristiansand
Rainfall       0     1          2          3             4
Humidity       5     6          7          8             9
Wind          10    11         12         13            14
```

[16]:
```python
stacked = dframe1.stack()
stacked
```

[16]:
```
Rainfall  Bergen           0
          Oslo             1
          Trondheim        2
          Stavanger        3
          Kristiansand     4
Humidity  Bergen           5
          Oslo             6
          Trondheim        7
          Stavanger        8
          Kristiansand     9
Wind      Bergen          10
```

```
           Oslo            11
           Trondheim       12
           Stavanger       13
           Kristiansand    14
dtype: int64
```

[17]:
```
stacked.unstack()
```

[17]:
```
          Bergen  Oslo  Trondheim  Stavanger  Kristiansand
Rainfall       0     1          2          3             4
Humidity       5     6          7          8             9
Wind          10    11         12         13            14
```

[18]:
```
series1 = pd.Series([000, 111, 222, 333], index=['zeros','ones', 'twos',␣
 ↪'threes'])
series2 = pd.Series([444, 555, 666], index=['fours', 'fives', 'sixs'])

frame2 = pd.concat([series1, series2], keys=['Number1', 'Number2'])
frame2.unstack()
```

[18]:
```
         fives  fours   ones   sixs  threes   twos  zeros
Number1    NaN    NaN  111.0    NaN   333.0  222.0    0.0
Number2  555.0  444.0    NaN  666.0     NaN    NaN    NaN
```

3.9 Forward and backward filling of the missing values

[49]:
```
dfx.store4.fillna(method='ffill')
```

```
/tmp/ipykernel_44487/4057730406.py:1: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
instead.
  dfx.store4.fillna(method='ffill')
```

[49]:
```
apple         20.0
banana        20.0
kiwi          20.0
grapes        20.0
mango         20.0
watermelon    18.0
oranges       18.0
Name: store4, dtype: float64
```

[50]:
```
dfx.store4.fillna(method='bfill')
```

```
/tmp/ipykernel_44487/1219575873.py:1: FutureWarning: Series.fillna with 'method'
is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill()
```

```
instead.
  dfx.store4.fillna(method='bfill')
```

[50]:
```
apple         20.0
banana        18.0
kiwi          18.0
grapes        18.0
mango         18.0
watermelon    18.0
oranges        NaN
Name: store4, dtype: float64
```

3.10 Filling with index labels

[51]:
```
to_fill = pd.Series([14, 23, 12], index=['apple', 'mango', 'oranges'])
to_fill
```

[51]:
```
apple      14
mango      23
oranges    12
dtype: int64
```

[52]:
```
dfx.store4.fillna(to_fill)
```

[52]:
```
apple         20.0
banana         NaN
kiwi           NaN
grapes         NaN
mango         23.0
watermelon    18.0
oranges       12.0
Name: store4, dtype: float64
```

[53]:
```
dfx.fillna(dfx.mean())
```

[53]:

	store1	store2	store3	store4	store5
apple	15.0	16.0	17.0	20.0	NaN
banana	18.0	19.0	20.0	19.0	NaN
kiwi	21.0	22.0	23.0	19.0	NaN
grapes	24.0	25.0	26.0	19.0	NaN
mango	27.0	28.0	29.0	19.0	NaN
watermelon	15.0	16.0	17.0	18.0	NaN
oranges	20.0	21.0	22.0	19.0	NaN

3.11 Interpolation of missing values

[54]:
```python
ser3 = pd.Series([100, np.nan, np.nan, np.nan, 292])
ser3.interpolate()
```

[54]:
```
0    100.0
1    148.0
2    196.0
3    244.0
4    292.0
dtype: float64
```

[55]:
```python
from datetime import datetime
ts = pd.Series([10, np.nan, np.nan, 9],
               index=[datetime(2019, 1,1),
                      datetime(2019, 2,1),
                      datetime(2019, 3,1),
                      datetime(2019, 5,1)])

ts
```

[55]:
```
2019-01-01    10.0
2019-02-01     NaN
2019-03-01     NaN
2019-05-01     9.0
dtype: float64
```

[56]:
```python
ts.interpolate()
```

[56]:
```
2019-01-01    10.000000
2019-02-01     9.666667
2019-03-01     9.333333
2019-05-01     9.000000
dtype: float64
```

[57]:
```python
ts.interpolate(method='time')
```

[57]:
```
2019-01-01    10.000000
2019-02-01     9.741667
2019-03-01     9.508333
2019-05-01     9.000000
dtype: float64
```

4 Discretization and binning

[61]:
```
import pandas as pd

height =  [120, 122, 125, 127, 121, 123, 137, 131, 161, 145, 141, 132]

bins = [118, 125, 135, 160, 200]

category = pd.cut(height, bins)

category
```

[61]:
```
[(118, 125], (118, 125], (118, 125], (125, 135], (118, 125], …, (125, 135],
(160, 200], (135, 160], (135, 160], (125, 135]]
Length: 12
Categories (4, interval[int64, right]): [(118, 125] < (125, 135] < (135, 160] <
(160, 200]]
```

[62]:
```
pd.value_counts(category)
```

```
/tmp/ipykernel_44487/2654243906.py:1: FutureWarning: pandas.value_counts is
deprecated and will be removed in a future version. Use
pd.Series(obj).value_counts() instead.
  pd.value_counts(category)
```

[62]:
```
(118, 125]    5
(125, 135]    3
(135, 160]    3
(160, 200]    1
Name: count, dtype: int64
```

[63]:
```
category2 = pd.cut(height, [118, 126, 136, 161, 200], right=False)

category2
```

[63]:
```
[[118, 126), [118, 126), [118, 126), [126, 136), [118, 126), …, [126, 136),
[161, 200), [136, 161), [136, 161), [126, 136)]
Length: 12
Categories (4, interval[int64, left]): [[118, 126) < [126, 136) < [136, 161) <
[161, 200)]
```

[64]:
```
bin_names = ['Short Height', 'Averge height', 'Good Height', 'Taller']
pd.cut(height, bins, labels=bin_names)
```

[64]:
```
['Short Height', 'Short Height', 'Short Height', 'Averge height', 'Short
Height', …, 'Averge height', 'Taller', 'Good Height', 'Good Height', 'Averge
height']
```

```
Length: 12
Categories (4, object): ['Short Height' < 'Averge height' < 'Good Height' <
'Taller']
```

[65]:
```
# Number of bins as integer
import numpy as np

pd.cut(np.random.rand(40), 5, precision=2)
```

[65]:
```
[(0.79, 0.99], (0.2, 0.4], (0.4, 0.59], (0.59, 0.79], (-0.00095, 0.2], …,
(0.4, 0.59], (0.4, 0.59], (-0.00095, 0.2], (0.59, 0.79], (-0.00095, 0.2]]
Length: 40
Categories (5, interval[float64, right]): [(-0.00095, 0.2] < (0.2, 0.4] < (0.4,
0.59] < (0.59, 0.79] < (0.79, 0.99]]
```

[66]:
```
randomNumbers = np.random.rand(2000)
category3 = pd.qcut(randomNumbers, 4) # cut into quartiles
category3
```

[66]:
```
[(0.503, 0.747], (0.252, 0.503], (0.503, 0.747], (-0.000865, 0.252], (0.747,
1.0], …, (-0.000865, 0.252], (-0.000865, 0.252], (0.747, 1.0], (-0.000865,
0.252], (-0.000865, 0.252]]
Length: 2000
Categories (4, interval[float64, right]): [(-0.000865, 0.252] < (0.252, 0.503] <
(0.503, 0.747] < (0.747, 1.0]]
```

[67]:
```
pd.value_counts(category3)
```

```
/tmp/ipykernel_44487/647498483.py:1: FutureWarning: pandas.value_counts is
deprecated and will be removed in a future version. Use
pd.Series(obj).value_counts() instead.
  pd.value_counts(category3)
```

[67]:
```
(-0.000865, 0.252]    500
(0.252, 0.503]        500
(0.503, 0.747]        500
(0.747, 1.0]          500
Name: count, dtype: int64
```

[68]:
```
pd.qcut(randomNumbers, [0, 0.3, 0.5, 0.7, 1.0])
```

[68]:
```
[(0.503, 0.699], (0.299, 0.503], (0.503, 0.699], (-0.000865, 0.299], (0.699,
1.0], …, (-0.000865, 0.299], (-0.000865, 0.299], (0.699, 1.0], (-0.000865,
0.299], (-0.000865, 0.299]]
Length: 2000
Categories (4, interval[float64, right]): [(-0.000865, 0.299] < (0.299, 0.503] <
(0.503, 0.699] < (0.699, 1.0]]
```

[69]:
```
df = pd.read_csv('https://raw.githubusercontent.com/PacktPublishing/
 ↪hands-on-exploratory-data-analysis-with-python/master/Chapter%204/sales.csv')
df.head(10)
```

[69]:
```
     Account            Company  Order               SKU        Country  Year  \
0  123456779          Kulas Inc  99985   s9-supercomputer          Aruba  1981
1  123456784             GitHub  99986   s4-supercomputer         Brazil  2001
2  123456782          Kulas Inc  99990  s10-supercomputer     Montserrat  1973
3  123456783          My SQ Man  99999   s1-supercomputer    El Salvador  2015
4  123456787         ABC Dogma  99996   s6-supercomputer         Poland  1970
5  123456778  Super Sexy Dingo  99996   s9-supercomputer     Costa Rica  2004
6  123456783         ABC Dogma  99981  s11-supercomputer          Spain  2006
7  123456785         ABC Dogma  99998   s9-supercomputer        Belarus  2015
8  123456778         Loolo INC  99997   s8-supercomputer      Mauritius  1999
9  123456775          Kulas Inc  99997   s7-supercomputer  French Guiana  2004

   Quantity  UnitPrice  transactionComplete
0      5148        545                False
1      3262        383                False
2      9119        407                 True
3      3097        615                False
4      3356         91                 True
5      2474        136                 True
6      4081        195                False
7      6576        603                False
8      2460         36                False
9      1831        664                 True
```

[70]:
```
df.describe()
```

[70]:
```
            Account         Order          Year      Quantity     UnitPrice
count  1.000000e+04  10000.000000  10000.000000  10000.000000  10000.000000
mean   1.234568e+08  99989.562900   1994.619800   4985.447300    355.866600
std    5.741156e+00      5.905551     14.432771   2868.949686    201.378478
min    1.234568e+08  99980.000000   1970.000000      0.000000     10.000000
25%    1.234568e+08  99985.000000   1982.000000   2505.750000    181.000000
50%    1.234568e+08  99990.000000   1995.000000   4994.000000    356.000000
75%    1.234568e+08  99995.000000   2007.000000   7451.500000    531.000000
max    1.234568e+08  99999.000000   2019.000000   9999.000000    700.000000
```

[71]:
```
# Find values in order that exceeded
df['TotalPrice'] = df['UnitPrice'] * df['Quantity']
df.head(10)
```

[71]:
```
     Account            Company  Order               SKU        Country  Year  \
0  123456779          Kulas Inc  99985   s9-supercomputer          Aruba  1981
1  123456784             GitHub  99986   s4-supercomputer         Brazil  2001
```

```
2  123456782         Kulas Inc  99990  s10-supercomputer     Montserrat  1973
3  123456783         My SQ Man  99999   s1-supercomputer    El Salvador  2015
4  123456787         ABC Dogma  99996   s6-supercomputer         Poland  1970
5  123456778  Super Sexy Dingo  99996   s9-supercomputer     Costa Rica  2004
6  123456783         ABC Dogma  99981  s11-supercomputer          Spain  2006
7  123456785         ABC Dogma  99998   s9-supercomputer        Belarus  2015
8  123456778         Loolo INC  99997   s8-supercomputer      Mauritius  1999
9  123456775         Kulas Inc  99997   s7-supercomputer  French Guiana  2004

   Quantity  UnitPrice  transactionComplete  TotalPrice
0      5148        545                False     2805660
1      3262        383                False     1249346
2      9119        407                 True     3711433
3      3097        615                False     1904655
4      3356         91                 True      305396
5      2474        136                 True      336464
6      4081        195                False      795795
7      6576        603                False     3965328
8      2460         36                False       88560
9      1831        664                 True     1215784
```

[72]:
```python
# Find transaction exceeded 3000000
TotalTransaction = df["TotalPrice"]
TotalTransaction[np.abs(TotalTransaction) > 3000000]
```

[72]:
```
2       3711433
7       3965328
13      4758900
15      5189372
17      3989325
         …
9977    3475824
9984    5251134
9987    5670420
9991    5735513
9996    3018490
Name: TotalPrice, Length: 2094, dtype: int64
```

[73]:
```python
df[np.abs(TotalTransaction) > 6741112]
```

[73]:
```
        Account     Company  Order                SKU       Country  Year  \
818   123456781   Gen Power  99991   s1-supercomputer  Burkina Faso  1985
1402  123456778    Will LLC  99985  s11-supercomputer       Austria  1990
2242  123456770     Name IT  99997   s9-supercomputer       Myanmar  1979
2876  123456772   Gen Power  99992  s10-supercomputer          Mali  2007
3210  123456782   Loolo INC  99991   s8-supercomputer        Kuwait  2006
3629  123456779   My SQ Man  99980   s3-supercomputer     Hong Kong  1994
```

```
7674  123456781  Loolo INC  99989   s6-supercomputer     Sri Lanka  1994
8645  123456789  Gen Power  99996  s11-supercomputer      Suriname  2005
8684  123456785  Gen Power  99989   s2-supercomputer         Kenya  2013

      Quantity  UnitPrice  transactionComplete  TotalPrice
818       9693        696                False     6746328
1402      9844        695                 True     6841580
2242      9804        692                False     6784368
2876      9935        679                False     6745865
3210      9886        692                False     6841112
3629      9694        700                False     6785800
7674      9882        691                False     6828462
8645      9742        699                False     6809658
8684      9805        694                False     6804670
```

5 Permunation and Random sampling

[74]:
```python
dat = np.arange(80).reshape(10,8)
df = pd.DataFrame(dat)

df
```

[74]:

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31
4	32	33	34	35	36	37	38	39
5	40	41	42	43	44	45	46	47
6	48	49	50	51	52	53	54	55
7	56	57	58	59	60	61	62	63
8	64	65	66	67	68	69	70	71
9	72	73	74	75	76	77	78	79

[75]:
```python
sampler = np.random.permutation(10)
sampler
```

[75]:
```
array([2, 8, 7, 6, 5, 9, 3, 1, 4, 0])
```

[76]:
```python
df.take(sampler)
```

[76]:

	0	1	2	3	4	5	6	7
2	16	17	18	19	20	21	22	23
8	64	65	66	67	68	69	70	71
7	56	57	58	59	60	61	62	63
6	48	49	50	51	52	53	54	55

```
5  40  41  42  43  44  45  46  47
9  72  73  74  75  76  77  78  79
3  24  25  26  27  28  29  30  31
1   8   9  10  11  12  13  14  15
4  32  33  34  35  36  37  38  39
0   0   1   2   3   4   5   6   7
```

[77]:
```
# Random sample without replacement

df.take(np.random.permutation(len(df))[:3])
```

[77]:
```
    0   1   2   3   4   5   6   7
5  40  41  42  43  44  45  46  47
9  72  73  74  75  76  77  78  79
3  24  25  26  27  28  29  30  31
```

[78]:
```
# Random sample with replacement
sack = np.array([4, 8, -2, 7, 5])
sampler = np.random.randint(0, len(sack), size = 10)
sampler
```

[78]:
```
array([3, 3, 3, 2, 1, 2, 1, 1, 0, 2])
```

[79]:
```
draw = sack.take(sampler)
draw
```

[79]:
```
array([ 7,  7,  7, -2,  8, -2,  8,  8,  4, -2])
```

6 Dummy variables

[80]:
```
df = pd.DataFrame({'gender': ['female', 'female', 'male', 'unknown', 'male',␣
↪'female'], 'votes': range(6, 12, 1)})
df
```

[80]:
```
    gender  votes
0   female      6
1   female      7
2     male      8
3  unknown      9
4     male     10
5   female     11
```

[81]:
```
pd.get_dummies(df['gender'])
```

[81]:
```
  female   male  unknown
0   True  False    False
```

```
1   True  False    False
2  False   True    False
3  False  False     True
4  False   True    False
5   True  False    False
```

[82]:
```
dummies = pd.get_dummies(df['gender'], prefix='gender')
dummies
```

[82]:

	gender_female	gender_male	gender_unknown
0	True	False	False
1	True	False	False
2	False	True	False
3	False	False	True
4	False	True	False
5	True	False	False

[83]:
```
with_dummy = df[['votes']].join(dummies)
with_dummy
```

[83]:

	votes	gender_female	gender_male	gender_unknown
0	6	True	False	False
1	7	True	False	False
2	8	False	True	False
3	9	False	False	True
4	10	False	True	False
5	11	True	False	False

6.1 Benefits of data transformation

1. Data transformation promotes interoperability between several applications. The main reason for creating a similar format and structure in the dataset is that it becomes compatible with other systems.
2. Comprehensibility for both humans and computers is improved when using better-organized data compared to messier data.
3. Data transformation ensures a higher degree of data quality and protects applications from several computational challenges such as null values, unexpected duplicates, and incorrect indexings, as well as incompatible structures or formats.
4. Data transformation ensures higher performance and scalability for modern analytical databases and dataframes.